NAVAL
POSTGRADUATE
SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**ANALYSIS OF CLOUD-BASED DATABASE SYSTEMS**

by

Matthew J. Clyman

June 2015

| | |
|---|---|
| Thesis Advisor: | Geoffrey G. Xie |
| Second Reader: | Arijit Das |

**Approved for public release; distribution is unlimited**

| **REPORT DOCUMENTATION PAGE** | *Form Approved OMB No. 0704–0188* |
|---|---|

| | | |
|---|---|---|
| **1. AGENCY USE ONLY *(Leave blank)*** | **2. REPORT DATE** June 2015 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** ANALYSIS OF CLOUD-BASED DATABASE SYSTEMS | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Matthew J. Clyman | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** |

**11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release;distribution is unlimited | **12b. DISTRIBUTION CODE** |

**13. ABSTRACT (maximum 200 words)**

To take advantage of cloud computing benefits that boost an enterprise's efficiency, innovation, and cost savings, the Department of Defense's (DOD) cloud computing strategy needs to evaluate databases as a service. If the DOD is going to prioritize outsourced database server hosting, each application's performance and agility of each must be assessed to determine if they can thrive in this new environment.

We performed an experiment to compare the performance between a current Naval Postgraduate School standalone database server and a cloud version developed specifically for the experiment. The cloud environment was created both with resources less equal to and greater than the live standalone server. We simulated cloud environment traffic based on the type of queries observed in production and collected data to compare its performance against the standalone database.

The results show that the cloud database performed similarly to or better than our standalone server, with equivalent resources. It achieved this level of performance without utilizing additional resources. We increased the resources dedicated to our cloud environment to test scalability, and we witnessed that the time needed to execute queries decreased significantly. We therefore concluded that our database would perform and scale favorably in a cloud environment.

| | |
|---|---|
| **14. SUBJECT TERMS** Database, cloud database, private cloud, performance analysis, SQL Profiler, Performance Monitor, PerfMon | **15. NUMBER OF PAGES** 65 |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UU |

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

# ANALYSIS OF CLOUD-BASED DATABASE SYSTEMS

Matthew J. Clyman
Civilian, Department of the Navy
B.S., California Polytechnic State University San Luis Obispo, 2009

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2015**

| | |
|---|---|
| Author: | Matthew J. Clyman |
| Approved by: | Dr. Geoffrey G. Xie<br>Thesis Advisor |
| | Arijit Das<br>Second Reader |
| | Dr. Peter J. Denning<br>Chair, Department of Computer Science |

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

To take advantage of cloud computing benefits that boost an enterprise's efficiency, innovation, and cost savings, the Department of Defense's (DOD) cloud computing strategy needs to evaluate databases as a service. If the DOD is going to prioritize outsourced database server hosting, each application's performance and agility of each must be assessed to determine if they can thrive in this new environment.

We performed an experiment to compare the performance between a current Naval Postgraduate School standalone database server and a cloud version developed specifically for the experiment. The cloud environment was created both with resources less equal to and greater than the live standalone server. We simulated cloud environment traffic based on the type of queries observed in production and collected data to compare its performance against the standalone database.

The results show that the cloud database performed similarly to or better than our standalone server, with equivalent resources. It achieved this level of performance without utilizing additional resources. We increased the resources dedicated to our cloud environment to test scalability, and we witnessed that the time needed to execute queries decreased significantly. We therefore concluded that our database would perform and scale favorably in a cloud environment.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| ACID | Atomicity, Consistency, Isolation, and Durability |
| DBaaS | Database as a Service |
| DBA | Database Administrator |
| DOD | Department of Defense |
| DMV | Dynamic Management Views |
| EU | European Union |
| GUI | Graphical User Interface |
| GUID | Globally Unique ID |
| IaaS | Infrastructure as a Service |
| MMC | Microsoft Management Console |
| NIST | National Institute of Standards and Technology |
| NPS | Naval Postgraduate School |
| PaaS | Platform as a Service |
| PERFMON | Performance Monitor |
| PYTHON | Program Yet to Have Original Name |
| RDBMS | Relational Database Management Software |
| SaaS | Software as a Service |
| SAN | Storage Area Network |
| SMO | Server Management Object |
| SQL | Structured Query Language |
| VM | Virtual Machine |

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

Private database cloud services or database as a server is not a cutting-edge idea. It has spent its time going through the adoption cycle and is now the accepted standard for development and testing environments both in the private and public cloud. Efforts have been made in the Department of Defense (DOD) to move toward a more cloud-based hosting model to take advantage of cloud technology, but there are few case studies documenting the results of these efforts. Our experiment sought to address this lack of data by migrating a current DOD production database to a cloud environment.

For this thesis we deployed and modified the resources available to a Virtual Machine (VM) version of the Naval Postgraduate School (NPS) PYTHON database server in the private cloud environment and evaluated its performance against the current standalone production database. The private-cloud resources granted to the VM were in configurations of 2 CPUs and 4 Gigabytes (GB) of RAM, 4 CPUs and 8GB, 8 CPUs and 16GB, and lastly 16 CPUs and 32GB of RAM. After deploying the VM, we installed SQL Server 2014 relational database management software (RDBMS) and restored a copy of the PYTHON database onto the server. To properly assess the performance in each configuration, a comparable traffic load to production needed to be generated. By examining the dynamic management views within SQL Server, we retrieved lists of the most commonly executed queries, the percentage of reads versus writes, as well as volume of each main query type (SELECT, INSERT, UPDATE, and DELETE.) This became the synthetic traffic load that we would run against each VM configuration so we could gather various performance metrics. This load was virtually identical when executed on our private-cloud VM, and was comparable to that experienced on our live production server. We then this synthetic traffic on our VM for 24 hours and repeated the process again for each different resource level. While the traffic flowed, we captured performance metrics using SQL Profiler and Windows Performance Monitor.

After analyzing the performance figures, we were happy to discover that our cloud-based VM, with the same resources as production, performed similarly. The time it took for a query to complete on average for the production system was 136,746

microseconds. On our cloud-based system, the average was 198,875 microseconds, executing only .062 seconds slower. The cloud-based VM was able to perform the same amount of work without utilizing a higher percentage of its CPU cores. We found our cloud-based VM would use just 2.97% of its four cores on average, compared to the production system's average of 5.36%. When we allocated even more processing power to the later trials at 8 and 16 CPU cores, we saw even greater performance gains. The 8-Core iteration averaged 124,478 microseconds for queries to complete, and 29,171 microseconds when repeated using the 16-Core iteration.

The results showed that our production database application is well suited for deployment in a cloud-based environment. The low numbers for the average processor usage showed that additional processing overhead is not needed for the database to function. In other words, no significant additional resources are required to obtain a similar level of performance in the private cloud. The server proved to scale appropriately with additional resources, proving that our database can take advantage of the elasticity of the cloud environment.

# ACKNOWLEDGMENTS

The author wishes to acknowledge the following people for their contributions to his thesis: Geoffrey G. Xie, Robert Beverly, Brent Ozar, Gregory Syme, Edgar Mendoza, and Kristen Smith.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

Cloud computing has become one of the most popular tech-buzzwords of the past decade. It has garnered attention from accountants and management for advertised cost advantages, and from system administrators and data analysts for its reliability and scalability. In an era of consolidated and centralized resources, both in the government and in private industry, cost savings is an important priority. We are living under the looming shadow of continued sequestration, with the United States operating globally with a smaller budget in an increasingly dangerous world. The results have been the suspension of our ships at sea, our helicopters in the air, our training on the ground [1], and the pay of our civilian workforce [2].

The adoption of cloud practices has been anything but swift. Concerns about security and access have prevented most efforts in the Department of Defense (DOD) thus far, and they should. Our own government has seized cloud-hosted data of European Union (EU) citizens under the Patriot Act [3]. Unforeseen virtualization bugs have caused wide-reaching outages [4], leaving customers helpless to assist. When errors happen, there is little companies can do but ask for updates from their service providers.

We are in an age in which the leaking of our private data and credit card numbers are not a possibility but an eventuality [5]. Why, in the face of all this, should we put our data and security in the hands of another when we have been storing it within our own firewalls for decades? The answer is cost savings and scalability.

Enterprises need to consider the benefits and the drawbacks of adopting cloud computing in their organizations. While apparent cost savings are the main impetus for adopting a cloud-computing model, the security, flexibility and compatibility of an organization's systems need to be maintained; otherwise, the new cloud system would degrade these properties. The storage of data, hosting of applications and performance of database systems all have to be individually evaluated and gradually migrated if cloud computing has any hope of being successful.

## A. PROBLEM STATEMENT AND METHODOLOGY

Few documented case studies focused on moving DOD database applications to cloud-based hosting exist. This experiment's goals were to move a DOD database to a cloud environment in order to understand the performance implications of the migration, and determine if our database performance scales with additional resources.

To accomplish this, we deploy a cloud-based version of a production database at the Naval Postgraduate School (NPS) and then create synthetic traffic based on the query composition of queries executed on the current production system. This load allowed us to accurately compare performance metrics from our VM environment. We looked at the average query completion time, resource utilization, and queues for resources.

Collecting data in such a manner required a few tools. First, to collect data regarding server performance, we used a Microsoft Windows tool called Performance Monitor. This gave us data regarding resource utilization and queueing. The second tool we used was the SQL Server Profiler provided by Microsoft SQL Server. This tool allowed us to capture a large array of SQL specific metrics. We were most interested in the average completion time. This was our main metric used for comparison.

## B. RESEARCH DESCRIPTION AND HYPOTHESIS

We hypothesized that we could identify the performance of our database if it were to be deployed in a cloud environment by gathering multiple performance metrics. Traffic can be gathered against our current system and then replicated in a test environment; this will allow us to gather a second round of metrics and compare them to those taken against our currently running database. We pose the following as our hypotheses, in order of importance:

1. The average query completion time for the private-cloud database will be similar to that of the standalone server when provisioned with a similar amount of processing and storage capacity.
2. The private-cloud environment will utilize the same amount of processing power or more in order to produce the same amount of throughput as the standalone server.

3. With increased processing and storage resources, we will see a minor increase in the performance of the private-cloud database.

## C. ORGANIZATION

Chapter II reviews the background information for the research. There are several sections which review the history and current state of cloud computing and outline our reasoning behind pursuing this research. This chapter sets up the main goal of the research topic, and sets up the reader to be able to understand our approach in the subsequent chapters

Chapter III uses the concepts and knowledge provided in Chapter II as a starting point for introducing our analysis strategy. We will review multiple strategies for analyzing SQL traffic and the resulting load on system resources and the effect on query throughput. Lastly, we develop a method for collecting data from a live system and review its counterpart in our test system.

The results of our data collection and analysis are presented in Chapter IV. We look at Performance Monitor data first. Secondly, we analyze data collected from SQL Server Profiler traces. We analyze the trace results captured from our test bed both before and after increasing system resources.

Chapter V includes the conclusions of the research and recommendations. We discuss the benefits and planned future work.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. BACKGROUND AND RELATED WORK

"Cloud computing" refers to the pooling of networking resources, either publicly or privately available, in order for a user to utilize a service or program without the required software or associated hardware installed on his or her workstation. The "cloud" refers to the almost nebulous location where this software and hardware is effectively located. Exactly where the software and hardware is located is not important for the end-user; all that is important is that services remain available whenever they are required.

Cloud computing is attractive to consumers for multiple reasons. There exists the illusion of a near-infinite amount of computing resources, which eliminates the need for users to plan ahead in their provisioning as long as the application can scale elastically. The ability to pay as you go reduces upfront investment and unnecessary overhead, because the amount of resources needed can scale back down during periods of decreased activity.

Many models of Cloud-based services exist; the one we are most interested in is databases as a service, but before we get into that we must first examine the origins of this technology.

## A. ORIGINS AND FUNDAMENTAL KNOWLEDGE

In the 1990s, adoption of Internet use became universally available [6]. Services such as mail, news, fora, and eventually entertainment were being provided over the Internet. One of the earliest examples of cloud computing that most people are familiar with is web-based email.

Back in 1995, access to email servers required that software be installed on each user's machine. AOL.com utilized the model shown in Figure 1 at that time [7].

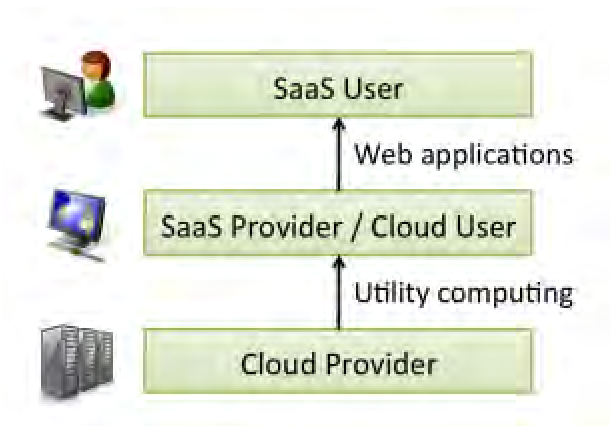


Figure 1. Cloud providers and their users. It is important to note that the SaaS Provider and SaaS Users could be one and the same. For example, business review site Yelp.com could be providing maps to businesses generated from a Google maps service.

At the end of the .com boom of the late 1990s, the majority of datacenters were utilizing 10% of their system resources at any one time [8]. This was due to the inflexibility of server provisioning. Successful service providers had to allocate enough resources to allow their system to still be available during peak usage. In times of reduced usage, there was no way to reduce power usage or hardware maintenance costs. You could not simply send the extra cashiers home and dim the lights. The booths stayed manned, the lights stayed on, and wood never stopped being fed into the fire.

In 1996, Hotmail and other web-based email cloud services emerged. For the first time, users could access their mail via a web browser without being responsible for where the software or hardware required for handling all that information resided [9]. With the expanse of computer utility servicing in the early 2000s, clients could enjoy greatly simplified software installation and maintenance while not being responsible for upgrading their own hardware.

In the early 2000s Intel, Amazon, and Microsoft provided these utility services, as well as datacenters hosting servers for VM deployment [10]. Now we had service suites including storage, computation and various other services.

It was not until October 2006, when Google released Google Docs and Sheets (a reworking of Writely), that software was offered as a service office suite. In Google Docs

[10], documents, presentations, and spreadsheets could be created, imported, or shared via email. Version history could also be saved on Google's servers.

## B. OVERVIEW OF CLOUD COMPUTING

Cloud computing is effectively the sharing of network resources to achieve a reduction in operating costs while simultaneously leveraging aggregated resources to greatly increase profits and coverage. Cloud computing is based around sharing services supported on a larger installed infrastructure.

Cloud computing is focused on maximizing the effectiveness of resources. "The Cloud" refers to a set of resources that are widely distributed while the underlying machinations are blurred, much like the haziness of a cloud. One of the key features of a cloud environment is the ability of the resources granted to a customer to dynamically grow to meet demand. Thus, regardless of the distribution of queueing for resources, the customer can efficiently grow his resource pool to meet demands in the short term, and dynamically shrink back to minimal levels to reduce costs [11].

In the same way that gas, water, and electricity are resources that can be accessed on demand, such is the goal of cloud-based computational power. The National Institute of Standards and Technology's (NIST) definition on cloud computing is as follows:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. [12]

Wikipedia, however, defines cloud computing as "deploying groups of remote servers and software networks that allow centralized data storage and online access to computer services or resources….classified as public, private or hybrid" [13].

Knowing what we do about cloud computing and data centers, we can safely say that cloud computing involves a centralized pool of resources that can be assigned to users nearly autonomously and then re-appropriated when not in use. The closer it resembles a utility service provider, with the ability to be monitored and access regulated, the more a service resembles a cloud computing platform.

## C. TYPES OF CLOUD ARCHITECTURES

Cloud computing is based upon three major types of provided services: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

### (1) Software as a Service (SaaS)

Software as a Service refers to applications being hosted by a service over the Internet. Most of the time these applications can be accessed with nothing outside of a web browser and account access. The user has no interaction with the server configuration or maintenance. An example of this would be email hosting like Gmail.

### (2) Platform as a Service (PaaS)

This portion of services allows customers to develop web applications without any of the complexity of creating and maintaining the underlying infrastructure. The standard PaaS is delivered with the provider handling the network, server, and storage as the host. The consumer installs and maintains the software configuration and upgrades. By software we mean that which the consumer has been developing, the underlying software needed for the hosting is all managed by the PaaS.

Database services fall under the PaaS umbrella [14]. A typical web application will have an application server, web server and database server, all of which would be managed by the PaaS, with the consumer only developing and deploying its application code.

### (3) Infrastructure as a Service (IaaS)

The third type of service provided is Infrastructure as a Service. The customer outsources the equipment used to support his operations to a service provider. This includes the servers, hardware, storage, and networking. The service provider is responsible for maintaining and updating the equipment [15].

## D. TYPES OF CLOUDS

There are numerous cloud deployment models; however, for the sake of brevity, this thesis only reviews the broader models addressed for evaluation.

#### (1) Private Cloud

The private cloud is an environment created and operated solely for a single organization. The management and hosting of resources can be handled either internally or externally. The main benefit of a private cloud is that security of data and access is more transparent in this method. Instead of segregating individual systems and servers, they all draw from the same pool, hoping to minimalize underutilized resources. This model still requires the company to build and maintain a datacenter as well as manage the virtualized environment for distributing resources. Because of this, the economic benefit is not as significant, but benefits can still be had.

#### (2) Public Cloud

The public cloud is an environment which hosts services available for public consumption. Services are offered on a pay-per-usage method. Security concerns are a bigger issue under this model as the company does not manage administrative access and simply has to trust that the datacenter is properly handling access control. It is the responsibility of the company to only be storing public-facing data in the cloud for consumption, but business processes are not always that straightforward.

#### (3) Hybrid Cloud

The hybrid cloud is a composition of two or more clouds [16]. This would allow an entity to have a section of their services managed privately on their own systems for security reasons. It would also allow it to utilize more cost saving measures by keeping its public facing content on a public cloud environment.

## E. CLOUD DATABASES BACKGROUND

A cloud database is simply a database that runs on a cloud computing platform. This could be from Microsoft Azure [17], Amazon EC2 [18], Rackspace, or others. The user can either upload or utilize a virtual machine template image for their database or they can purchase access to a database as a service (DBaaS). Both options allow for elastic scaling of resources to meet demand requirements.

#### (1) Database as a Service (DBaaS)

Multiple cloud platforms offer database as a service, where a user is not responsible for launching or maintaining the database software. Under this model the company such as Microsoft would launch a database for the user and charge them by its usage. This model can be as limited as providing a client with a VM image with database software preinstalled, or the cloud provider could host and manage the database of one of its own replicated clusters.

#### (2) Scaling of Cloud Databases

Virtualization has enormous benefits for cloud computing, namely databases. However, running a database in a virtual machine is not the same as database virtualization. In a dedicated server environment, a spike of utilization would have to be handled by the existing resources available to the created VM.

## F. BENEFITS OF CLOUD COMPUTING

There are a myriad of advantages to cloud computing. The most commonly discussed are the reduction in initial investment, scalability of environment, increased availability and reduced points of failure. Besides these there are even more subtle advantages such as smoother mergers and acquisition and flexibility to try out new technologies [19]. However, for this thesis we will only discuss the most common topics and briefly mention the subtle differences.

#### (1) Less Initial Investment

When deploying an application over a cloud service, very little investment is needed for infrastructure or hardware. In an organization that hosts all its servers in-house, resources would need to be initially allocated from its Storage Area Network (SAN) and deployed via a VM. If this organization does not use a SAN, or has maxed out its resources, a new server must be purchased before any work can begin. With a cloud deployment, little to no resources must be expended until the service is in use, and as the load increases beyond its allocated level of resources it can scale to meet the need.

### (2) Scalability

The rapid provisioning and release of resources is fundamental to cloud computing. If you ask anyone who has spent time in server provisioning you will hear the same complaint over and over; that allocated resources are under-utilized. Depending on the application, it may be dangerous to release CPU cores or decrease the RAM without modifying system parameters. None of this can be done when the system is online. When it comes to storage space the picture is even bleaker. Most monitoring software begins alerting administrators when capacity on a drive drops below a pre-set threshold. Having a storage buffer is the best scenario, meaning lots of under-utilized storage media. In a cloud environment you can add storage as you reach maximum capacity, not at the conception of your system.

### (3) Increased Availability

Two items affect availability; outages and downtime. In a replicated cloud environment, the odds of having a hardware failure affect one's environment is exceedingly low [20]. Depending on the requirements of your system even further precautions can be placed to minimize outages. Not readily apparent, but noteworthy, is the maintenance afforded by having one's server upgraded and patched by an experienced cloud team. This minimizes scheduled downtime, and can even be performed without taking one's application offline. Having a distributed cloud deployment increases the amount of points that must fail, thus increasing the resilience of one's environment.

### (4) No Single Points of Failure

The underlying resources of a deployed server traditionally all reside on the server blade plugged into a server rack. With cloud computing, the CPU's may span multiple blades, servers, and even amongst different data centers [21]. This also affects network bottlenecking. Now all the data that must be processed is flowing over multiple network pathways, reducing the chances of an oversaturated network resulting from an unexpected flux in traffic.

## G. DEPLOYMENT CONSIDERATIONS FOR CLOUD DATABASES

The maintenance of atomicity, consistency, isolation, and durability (ACID) is the main obstacle to implementing transactional cloud databases [22]. In a cloud database the state of data must remain consistent. The read/write operations of a database have to be executed without sacrificing ACID properties. If one user is attempting to read a record that is currently being updated by a concurrent user the traditional operation that occurs is a lock. The user reading the record must wait until the update transaction completes and is committed before he is allowed to read the record. As soon as multiple copies of the database are introduced among a single or multiple datacenters, the synchronization becomes a challenge. Atomicity requires that all operations of a transaction are done successfully or none of them are.

# III. OUTLINE OF EXPERIMENT

The goal of this thesis was to test a production database in a private cloud environment and analyze its performance. This was performed in a test environment, all traffic was simulated to be similar to the load experienced in production. In order to do this we first analyzed the current network traffic and differentiated the types of queries that made up our model.

Due to my position at NPS as the lead database administrator and my familiarity with the Identity Management System (IMS), I have a high-fidelity insight into the nature of the data collected. My approach was white-box, as I could see what volume of traffic was occurring and the exact makeup and queries performed.

This level of fidelity allowed me to build a more realistic picture of how performance was affected in the environment, since information other than job length and duration were known. Information such as tables queried and joins performed were made available.

I deployed our IMS database system in a private cloud environment. After setting the database, I created a load similar to that occurring on our production system. Live data was sampled to create a baseline from which analysis could be made and the technology evaluated.

## A. DESCRIPTION OF APPLICATION

For this experiment, we wanted to not just test that a standard database can be scaled effectively in a virtualized environment; it had to be one with which we could modify the load and verify that it fit the standard load profile. After reviewing the various systems and applications at NPS, we decided to utilize our IMS.

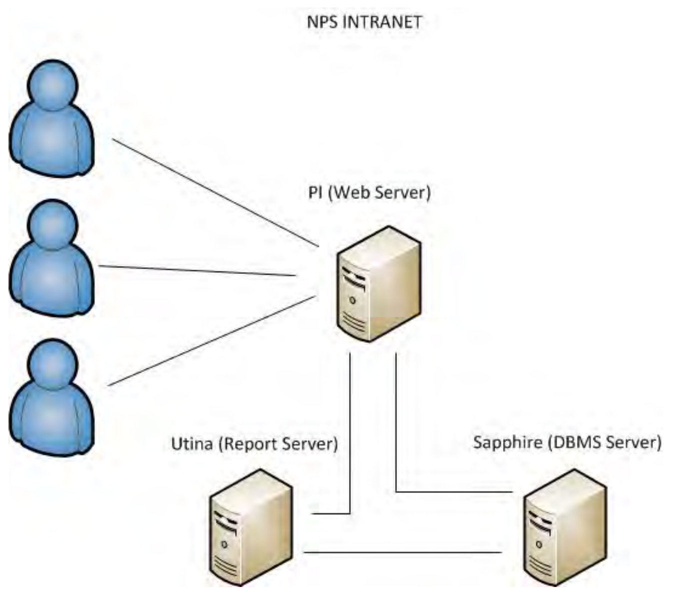


Figure 2. The users connect through the webserver PI, which connects to the internal database Sapphire. Utina is the reporting server that is indirectly accessed by the users.

Our IMS system at NPS was described by Registrar Director Mike Andersen as "a mission-critical system to NPS. If (when) [IMS] fails, education at NPS stops. There is no scheduling, no [grading], no transcripts, or no diplomas. [IMS] is essential to achieving the mission of the school." IMS is a SQL Server database currently installed on a Windows Server 2008 OS. The production system utilized 4 CPUs and 8GB of system RAM. Due to my relation with the datacenter administrators here on campus, getting a similar machine with fewer resources would prove an easy task.

## B. CREATING A BASELINE OF SYSTEM LOAD

In order for this experiment to be a success, we generated load as close to actual production traffic as possible. The environment did not permit a replicated server, so we gathered traffic with other methods. We utilized a combination of Performance Monitor traces and events captured by SQL Profiler.

### (1) Performance Monitor

Windows Performance Monitor (PerfMon) is a Microsoft Management Console (MMC) snap-in. It combines the functionality of other Microsoft tools including Performance Logs and alerts, Server Performance Advisor, and System Monitor [23]. It provides a graphical interface for customizing the collector sets as well as the events that were traced.

PerfMon can be used to identify hardware performance limitations and has hundreds of possible events that it can track. This includes but is not limited to: physical hard disk monitoring, memory allotment, utilization percentage of processors and the maximum throughput of the network interfaces. When gathering this data, PerfMon saves the data in a trace file that can be opened with the PerfMon GUI later and sorted.

The trace files we created were against the production database system during working hours. For the first performance monitoring session Performance Monitor was run using the following counters against our production database server for 48 hours. This would give us a solid baseline in terms of database query activity and hardware utilization.

When utilizing Performance Monitor, we captured a multitude of metrics. Clearly, information regarding the load on the processors and memory would need to be captured, but the inclusion of items such as disk writes and reads offer us some greater insight. When a server reaches its maximum allocated memory, it has to page to disk. Capturing data about disk writes in an environment where close to none should be occurring is clearly telling of its performance. The following are the metrics that we collected and an explanation of each counter that would be utilized.

### (2) Processor Queue Length

When a set of one or more threads is not able to run on the processor due to another active thread running we have a processor queue. The Processor Queue Length metric shows how many threads are in the queue and are unable to currently use the processor. A bottleneck occurring here is sign of a lack of resources and would have to be remedied by balancing the workload between computers or additional processors.

### (3) Batch Requests/Sec

Batch Requests represents the number of statements executed per second. When correlated with other metrics, especially CPU Usage, an overall understanding of SQL Server’s possible throughput can be learned.

### (4) Memory Grants Pending

Memory Grants Pending represents the current number of processes waiting for a workspace memory grant. This counter keeps track of the number of processes waiting for a memory grant to execute. An ideal number would be 0 for this metric.

### (5) User Connections

The User Connections counter identifies the number of different users that are connected at the time. This figure helps identify the usage schedule of one’s system, but must be correlated with other factors to evaluate the impact on one’s system.

### (6) Page Life Expectancy

Page Life Expectancy measures how long pages stay in memory in seconds. The longer a page stays in memory, the more likely SQL Server will be able to read the query results from memory instead of reading from disk.

### (7) Percentage Processor Time

The percentage of elapsed time a processor spends to execute a non-idle thread. It is calculated from measuring the percentage of time that the processor spends executing the idle thread and then subtracting that value from 100%.

### (8) Disk Reads/Sec

Performance Monitor captures the total number of individual disk IO requests completed over a period of one second. If the capture interval is set for anything greater than one second, the average of the values captured is presented.

#### (9) Disk Read Bytes/Sec

Performance Monitor captures the total number of bytes retrieved from the disk (read) over a period of time of a second. If the capture interval is set for anything greater that one second, the average of the values captured is presented.

#### (10) Disk Write Bytes/Sec

Performance Monitor captures the total number of bytes sent to disk (write) over a period of time of a second. If the capture interval is set for anything greater than one second, the average of the values captured is presented.

The load generated from the PerfMon trace is negligible, thus we did not factor it into the overall load of the server. The only consideration that we made was the growth of files generated during this process. After running a few smaller trace sessions we estimated that a whole day of continuous monitoring would only consume ~1GB of storage space; the data would be generated on a separate logging server, allowing any writing to disk to not impact daily query performance.

## C. SQL PROFILER TRACES

SQL Server Profiler is a server tool that lets a user capture and analyze events occurring within SQL Server [24]. The events could be everything from a remotely executing stored procedure or an administrator running an ad-hoc query.

At the most basic level, SQL Server Profiler is only a GUI that lets a user interface with another feature of SQL Server called SQL Trace. SQL Trace is responsible for doing all the heavy lifting when Profiler is capturing SQL Server events and storing them. SQL Trace can be accessed in multiple ways. The first way is indirectly using the Profiler GUI, as we are about to do. The second way is from using built in stored procedures, and third using Server Management Object (SMO).

Overall, SQL Trace is a simple communication monitoring tool. It functions similarly to network sniffers such as Wireshark that captures traffic on the network. The real difference is that SQL trace is more specialized in such that it captures traffic related to SQL Server and allows you to see the events occurring between client and SQL Server.

Additionally, unlike Wireshark, which captures every packet sent over a network, SQL Trace only captures and processes SQL Server events.

Firstly, a SQL Server event will occur between a client and the SQL Server itself. This could either be an application server gathering information to populate a report, or simply an administrator running an ad-hoc query. A wide range of events and information can be gathered in this manner. It is the job of SQL Trace, and the designer of the trace, to capture only the SQL Server events that are of interest and filter out those they have no use for.

Here is a small subset of items that SQL Profiler can help monitor:

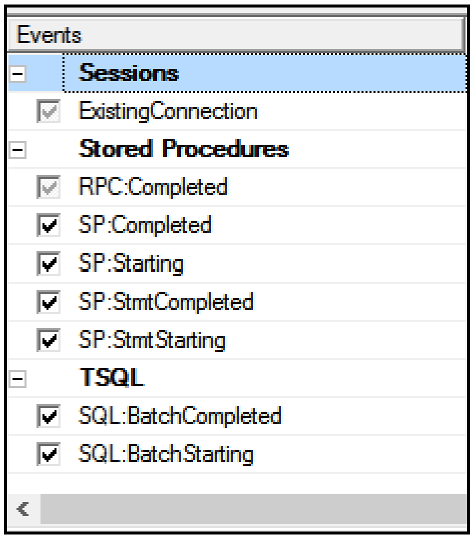
- Front-end application connections, queries, T-SQL, transactions
- Execution plan performance analysis
- SQL Server errors and warnings
- Traces of activity can be used to reproduce problems, can be saved and replayed
- Audit user activity
- Group or aggregate trace results for analysis
- Create custom traces
- Save traces to XML or CSV or to a Database table
- Perform stress testing

This is certainly not an exhaustive list, but it is enough to show that SQL Server Profiler was a great fit for our analysis job. We needed to be able to see exactly what queries were executed against the database, have enough fidelity to see if they were read/write/insert/update statements and also determine the average time it took to complete.

After the filter was applied, the data was queued in memory; henceforth, they could be saved in a file, a database table (both locally and remotely), or to an SMO-based application. As you can see from the figure above, SQL Profiler itself is in fact an SMO-based application used to access trace files. When all is said and done, the SQL Trace exists in a state which could not be directly accessed. One needed to interact indirectly using either an SMO-based application or a management tool able to write SQL queries.

For our research we decided to filter the SQL trace files into a remote database table, both to reduce congestion on our production and test machines and also to allow us to write SQL queries against the data for analysis.

We gathered data that displayed a detailed view of the load on our server. We needed a way to measure maximum and average query length as well as the actual makeup of queries. This included whether a query was a read/write/insert/update statement. To attain this, the following events were captured using SQL Profiler.



Events

- **Sessions**
  - ✓ ExistingConnection
- **Stored Procedures**
  - ✓ RPC:Completed
  - ✓ SP:Completed
  - ✓ SP:Starting
  - ✓ SP:StmtCompleted
  - ✓ SP:StmtStarting
- **TSQL**
  - ✓ SQL:BatchCompleted
  - ✓ SQL:BatchStarting

Figure 3. This trace shows information regarding the Application that connects, the username, loginname, reads, writes, the process ID, the start time, end time, and binary data which is the SQL being executed against the database.

The trace shown in Figure 3 was run against our production database over one complete working day. The data collected was representative of what an average workload would be for our application. All the data collected from this trace was inserted into a remote SQL Server database and into a table. This table could later be queried to gather the requested fidelity of information about our database. After gathering the data, we calculated the total updates, selects, inserts, the average processing time for each query, the standard deviation, and the duration. Having this allowed us to create a proper baseline from which to see if our testing environment will be able to perform similarly to production and thus we were able to tell if our experiment was delivering better performance. We would compare the data collected from both the SQL Profiler traces and the PerfMon events between both our cloud environment and our production environment to determine how our performance faired.

# IV. RESULTS AND ANALYSIS

We collected over 8.5 million rows of data from one whole day of traffic against our production database server. Our trace collected exactly which queries and procedures were executed during the business day. This included standard stored procedures executed by the EMS application and adhoc queries ran by administrators of the system. We collected data showing who ran the query, what time it began and when it ended. With this information we were able to build a standard distribution of query types and average operating values. From this data we could begin building a similar load on our cloud test-bed.

## A. DATA COLLECTION, PARSING, AND ORGANIZATION

Once we finished collecting the trace data, we knew we needed to have as close a recreation of data as possible. The obvious requirements were a consistent latency figure followed by the same percentage of query distribution. This would also be the same for the resource utilization percentages. By running a set of queries with random parameters attached, we could bypass caching employed by the database and get more accurate latency results. After establishing a baseline similar to that employed on our production machine we will increase the resources allocated to our test machine and analyze results. The hypothesis was that we should see at least some minor increase in the query performance and a decrease of resource utilization. As we doubled the amount of resources throughout the experiment the expectation was to see a series of increased performance results.

### 1. Data Collection

We used a combination of SQL Profiler traces, Performance Monitor logs and SQL Server Dynamic Management Views to collect data from both our production and testing environment. The data collected from the SQL Profiler traces were passed into a set of tables on a remote SQL Server database. The data collected would be pruned by a set of T-SQL scripts and then queries ran against the resulting data to gather statistics about query makeup and performance. We collected data from one full business day on our production system and similarly against our development system at two resource levels. Using Performance Monitor we gathered statistics over a range of metrics and outputted that data into excel spreadsheets. The data points from both levels of resource on our development machine and in production were compared and organized into graphs. Finally, data collected from the SQL Server Dynamic Management views gave us information regarding the breakup of types of queries in percentages. Overall the traces collected over 500,000,000 events between the five setups for analysis.

### 2. Reading SQL Profiler Data into SQL Table

Analyzing SQL Profiler data can prove problematic if one does not understand the methods available for accessing the said data. SQL Profiler information can be seen as a sort of black box, one that cannot be accessed unless you have compatible software or code that can access the SMO. In our case we utilized an idle SQL Server available for us to insert data as it was captured. This reduced the load on our test servers as nothing had to be written to the local disks. As each row of data was captured it was in turn written into a structured table in our remote SQL Server database. In this way we were able to query every line of data captured and begin to analyze the query makeup. Eventually we will have a single table for each SQL profiler capture: one for our initial production capture, a second for the low-resource test system, and a third for our test system with increased resources matching production. Following that, we ran two additional tests where we would double the resources. The fourth test would be using 8 CPU cores and 16GB of RAM, double that of the production system. The fifth test would utilize 16 CPU cores and 32GB of RAM, four times that of production.

### 3. Query Composition Gathering

In order to determine the breakdown of types of queries used, either reads or transactional queries, we utilized the SQL Server Dynamic Management Views (DMV). When pulling data from the sys.dm_io_virtual_file_stats DMV, we can get the query composition since the database was last restarted.

Table 1. The sys.dm_io_virtual_file_stats DMV produces data regarding the amount of reads/writes that have occurred since the database was last restarted. The number of reads/writes are in a magnitude of 1000s.

| # Reads | # Writes | # Bytes Read | # Bytes Written |
|---|---|---|---|
| 283130 | 30191 | 6283847270 | 742563840 |
| # Reads % | # Writes % | Read Bytes % | Written Bytes % |
| 90.4 | 9.6 | 98.8 | 1.2 |

Here we saw the amount of total data retrieved from these queries. What we were most interested in is the percentage of read queries as compared to the number of writes. Here we saw that since our database was last restarted a month ago, the percentage of reads was 90.4% and the writes numbered 9.6%. We compared these numbers to our SQL Profiler data and decided which figures we wanted to use for a baseline. Since our employee management system is primarily used as a reporting server, this large percentage of writes was due to ad-hoc administrative queries.

After loading all our data from SQL Profiler into our database we got the number of reads/writes/updates written during that day in production.

Table 2. The query makeup from a day in production.

| | Select | Insert | Update | Total | % Select |
|---|---|---|---|---|---|
| PROD | 3611133 | 11235 | 50871 | 3673239 | 98.31 |

Here the read percentage fitted more closely to what we determined was accurate. The data collected during our day in production now was used to create a baseline for query distribution that we applied to our test system.

## B. BASELINE MAKEUP OF PRODUCTION

Before beginning, we needed to have a close approximation of what types of queries are used in production. We then replicated those same queries, at the ratio collected in our previous steps on our private cloud test bed to simulate the same traffic.

In production the most popular type of query was a read query, unsurprisingly, since the EMS database primarily provides reports to students and staff. The percentage of reads it produces is roughly 98.31% of all total queries. That portion was not be difficult to reproduce, as one could just schedule read queries at a much higher rate than a set of reasonable inserts. To gather a better insight into the performance we decided to review the most common queries performed in production.

To create a similar assortment of queries as performed on our production system we had to go a few steps further than simply balancing out the read/write percentages to match. We wanted to match the same queries executed on production. To do this we queryed another DMV: sys.dm_exec_query_stats. Here we selected the top 10 most commonly executed queries. However, because this database serviced a reporting server, the majority of queries were executed using a parameter passed in at runtime. The simple response to this was to hard-code in a variable so that we could schedule this query. However, SQL Server would simply cache our query pages and never have to read from disk, thus skewing the data to preclude that our under-resourced test bed was performing faster than our beefier production system. To get around this we implemented a common SQL querying function.

## C. IMPLEMENTATION AND STACKING OF QUERIES

In order to prevent the database from simply caching all the queries we were looping, we had to get a little creative. To get around this we appended a function called NEWID to our queries. An example is:

```
Select top 20 percent *
From tnpemployee
Order by newid();
```

The function NEWID generates a Globally Unique ID (GUID) in memory for each row. By being a GUID, its number is unique and random. Thus, when the data was requested by GUID the results was a random set of data from the table. By performing our queries in this manner, we prevented the database management system from simply caching our queries in memory instead of utilizing system resources to retrieve the data, as if it was live data in a production system. After performing this on all of our system configurations we got an accurate readout of the query breakdown.

Table 3. The query makeup after running the top ten queries against our development machine using the `newid()` function for a whole day. This data was inserted using SQL Profiler into a table in our remote database and then sorted.

| | **Select** | **Insert** | **Update** | **Total** | **% Select** |
|---|---|---|---|---|---|
| **2-Core** | 3581965 | 1931 | 12140 | 3596036 | 99.6 |
| **Prod** | **3611133** | **11235** | **50871** | **3673239** | **98.3** |
| **4-Core** | 5969943 | 2703 | 16353 | 5988999 | 99.7 |
| **8-Core** | **7126641** | **3120** | **17545** | **7147306** | **99.71** |
| **16-Core** | 8167339 | 3345 | 17669 | 8188353 | 99.74 |

We saw the queries were properly distributed according to the query breakup among all the systems. We also saw that as we increased system resources, the queries completed quicker, and we'd process more traffic. In the case of the inserts and updates, we saw them peaking ~3300 as the server got closer to the amount of queries scheduled.

## D. DATA COLLECTION AND ANALYSIS

The queries replicated in our test environment are from a list of most common queries, which we were able to produce by querying the native Dynamic Management Views (DMV) built into SQL Server. All the data generated from our SQL Server Profiler traces was outputted into a remote database and then queried the resulting dataset for throughput figures.

Table 4. The average and maximum duration of query times as well as the average and max number of reads per request for each system.

| | Query Completion Time (microseconds) | | | Reads | | |
|---|---|---|---|---|---|---|
| | Avg | Max | Std Dev | Avg | Max | Std Dev |
| 2-CORE | 492559 | 11022718463 | 33183575 | 414 | 1925467 | 13430 |
| PROD | 136746 | 69918999 | 1325329 | 787 | 25385963 | 5041 |
| 4-CORE | 198875 | 19775131 | 1317404 | 270 | 491522 | 30153 |
| 8-CORE | 124478 | 11838789 | 212562 | 104 | 1916674 | 3347 |
| 16-CORE | 29171 | 1436325 | 53993 | 22 | 1916674 | 716 |

When running the same queries, our 2-Core machine hosted in our private cloud environment averaged almost 3000 extra microseconds to complete its set of queries. Also the most expensive query that queries our biggest table in the database took 157 times longer to complete in our development system. On our production system it took slightly longer than a minute to complete, and over 3 hours to complete in development. When we reviewed the standard deviation and variance, we saw the differences a bit more clearly.

For a query on our lower-scaled cloud database, the standard deviation was roughly 33 seconds, with an average query completion time of .49 seconds. In production the average query completion time was much faster, at .13 seconds. The standard deviation was only 1.3 seconds as well.

First off the duration of our cloud database was slightly higher than that of production, but not by much more than 200ms. Most shocking though was the time required to complete the most taxing query. On our production server it took 69.9 seconds and in the cloud just under 20. Again the standard deviation for our comparable cloud database performs within 8ms of that of production. Now when resources are increased we saw that same query completion time diminish again. With 8 cores the query took 11.8 seconds. With 16 cores we got down to 1.43 seconds.

The explanation for why the production max duration query length was so high comparatively is because the query executed in production was adhoc query performed by an administrator during a period of heavy congestion. The other max duration queries for the other four systems were the same. This information was chosen to be reported so that we can see the overall max duration decrease among our cloud systems. Now we turn our attention to figures regarding the reads reported for each server configuration.

Reads is the amount of logical disk reads committed on behalf of the server. These are logical, not physical disk reads, which happen when data requested is not cached in memory. If we are examining how fast one query would take between the five systems at load we would have to compare the CPU and MEMORY resources required as well as the logical reads. In certain cases, the database system may find it easier perform a nested loops operator, retrieving the pointers to all relevant rows then performing a key lookup to retrieve columns for the select list would be more efficient. Now, because each row had to be ‘index seeked’ individually, the same pages needed to be accessed in memory multiple times, each counting as a logical read. As a result, the total number of logical reads increased significantly.

The read data collected did not allow us to reach many conclusions. The low throughput of the slowest machine attributed to a low average amount of reads posting to the disk, but when increased to match production we saw nearly five times the reads. Again there are many interpretations of the data. Now that we have much better resourced machines for our final iterations we can clearly see that the amount of reads drastically diminish, because the pages read from memory needed to be retrieved less times for each query due to more available processing power.

## E. PERFORMANCE MONITORING RESULTS

Using Performance Monitor we measured and collected information. Each metric was important to understanding the overall health and throughput of the system evaluated. The following counters were examined over the course of a whole day:

- Processor Queue Length
- Batch Requests/sec
- Memory Grants Pending
- User Connections
- Page life expectancy
- Percentage Processor Time
- Disk Reads/sec

- Disk Read Bytes/sec
- Disk Write Bytes/sec

We overlaid the data collected from production with the lower and upper bounded cloud databases.

### (1) Processor Queue Length

The graphs in Figures 4 and 5 compared processor queue lengths among the different production variables.

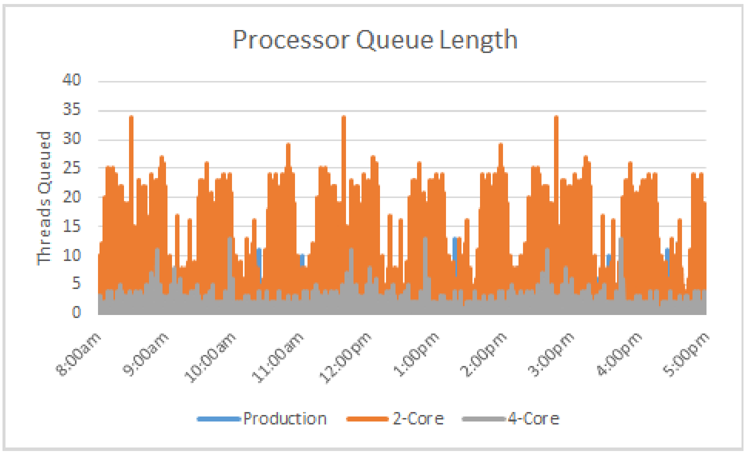


Figure 4. The queuing compared between production, and our first two server iterations with resources at half than and equal to production.

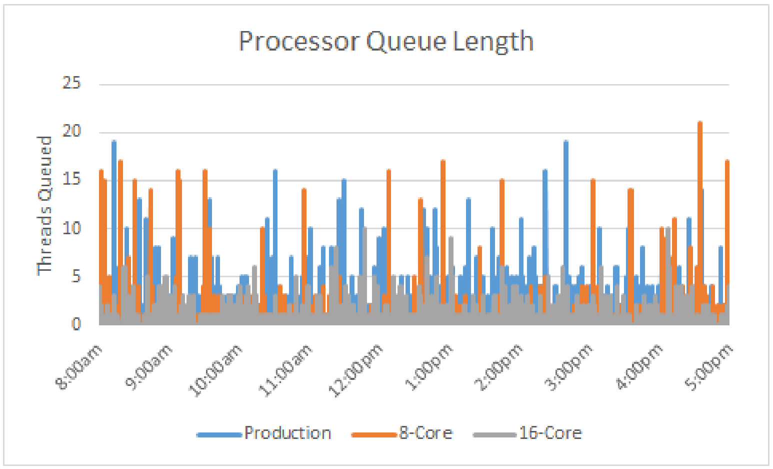


Figure 5. Comparison of our production machine to the 8-Core and 16-Core test machines.

The values in Table 5 show the average results for the amount of threads queued on the five iterations of tests. We saw that the queue is similar for our cloud server with equal resources as production. Otherwise the queuing appeared as we expected. As more cores were added the queueing dropped dramatically. As we got past 8 cores, there was only a slight amount of additional performance to be gained from upping the CPU cores.

Table 5. Average processor queue performance for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 3.9 | 0.089 | 0.11 | 0.036 | 0.032 | 0.8334 |

**(2) Batch Requests**

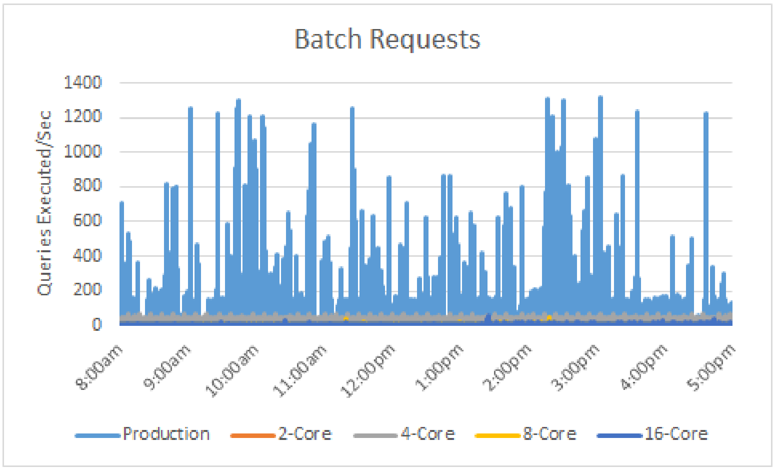


Figure 6. Batch requests of our 5 test iterations overlaid in a line chart.

Table 6. Average batch requests for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 0.3 | 1.21 | 17 | 0.89 | 1.56 | 4.192 |

Batch requests represent the number of statements executed per second. In other words it was a measurement of the database’s throughput. If the server was not under significant load than a lower batch requests figure is not noteworthy. However, as we could see in the previous graph for our lesser resourced database, there was a high queue to use the CPU. Thus, we could conclude that there was significant load, but the throughput was not sufficient to handle it. Now when we examined the greater-resourced databases, that same load was not significant enough for us to gather meaningful results from this metric. There simply was not enough volume presented for our final two iterations to test its maximum throughput here.

#### (3) Memory Grants Pending

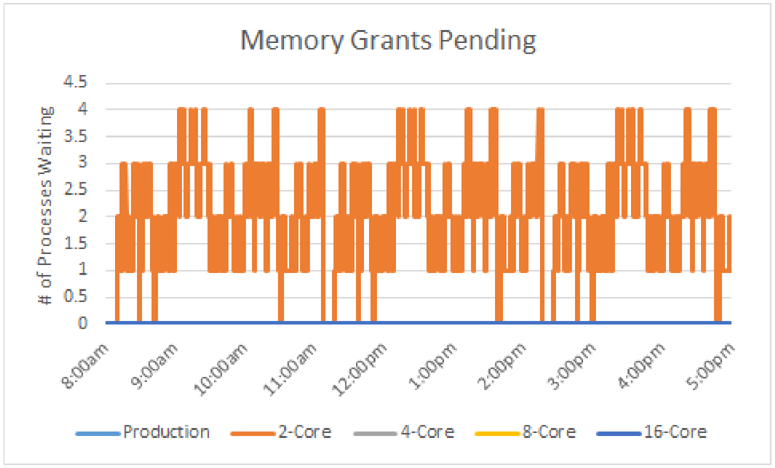


Figure 7. Memory Grants Pending for our 5 iterations. The only server that saw any grants pending was our first iteration with 2 cores.

Table 7. Average memory grants pending for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 2.04 | 0 | 0 | 0 | 0 | 0.4 |

Memory grants pending represents the number of processes waiting for a memory grant to be able to execute. The ideal number for this metric is 0, which we obtained in our production and higher-resourced databases. The takeaway from this metric collection is that the performance of the distributed memory in the cloud environment performed sufficiently as in there was no noticeable slowdown due to it being more distributed.

### (4) User Connections

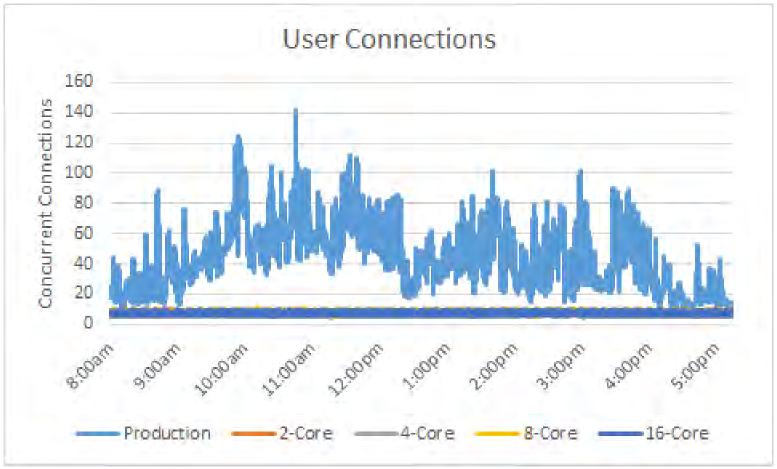


Figure 8. The concurrent user connections for each experiment.

Table 8. Average amount of user connections over the 9-hour window for each system.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 8.5 | 6.45 | 44.81 | 7.7 | 7.27 | 14.9 |

User connections measures the number of different users connected concurrently at any point in time. The question arose; why were there more users connected to our lesser resourced server than the greater resourced server? Both servers have exactly the same set of scheduled jobs executing at exactly the same time. The answer is that on the lesser resourced server, the time it took to complete a query is on average ~10000 microseconds or .01 seconds longer. This means that due to previously scheduled jobs queued up to execute, additional connections from the task scheduler were required. In production you can see that, throughout the day, we get on average 44 concurrent connections from staff and students. This metric only serves to display usage patterns for our actual production system; we gain scarce information about our test systems which all are using procedurally generated queries.

### (5) Page Life Expectancy

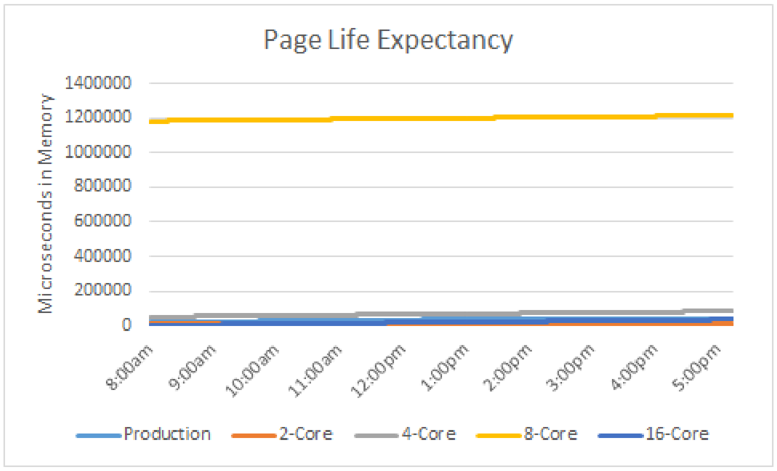


Figure 9. Page life expectancy of queries in memory.

Table 9. Page life expectancy for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 11257 | 65699 | 31946 | 1200582 | 17661 | 265429 |

Page life expectancy measures how long pages stay in memory is seconds. The longer a page stays in memory, the more likely SQL Server will be able to read the query results from memory instead of disk. Overall this metric is rather inconsequential as every new query is stored in memory when possible. The reason that the value for the test system had a much higher value is that the cloud database was left online for a few weeks before the resources were increased and measured; thus the server was online and had more queries scheduled against it, thus ticking up the page life values higher and higher and never maxing out due to such a high amount of allocated memory. Had we had a more memory intensive set of queries, we might see something other than an iterating line.

## (6) Percentage of Processor Time

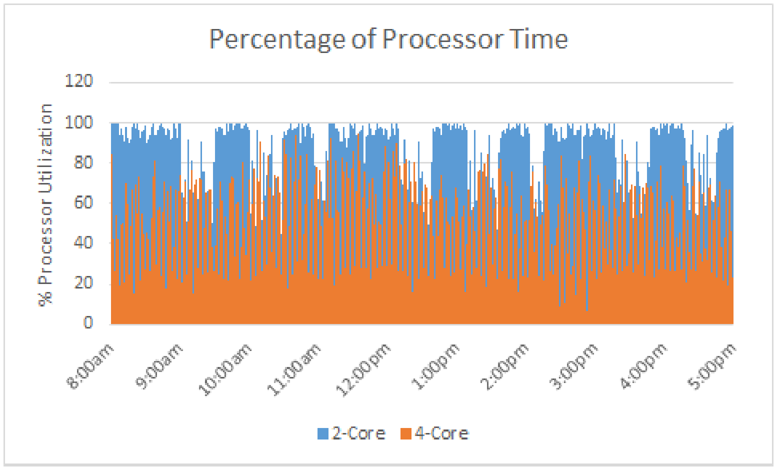


Figure 10. The percentage of processor utilization for our first two system iterations overlaid.

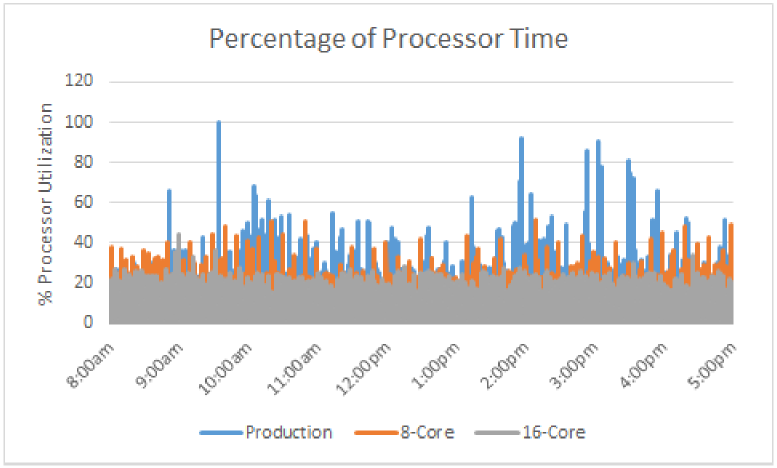


Figure 11. The percentage of processor utilization for our production system and our two higher resourced systems.

Table 10. Average % usage of the processor being utilized for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 68.97 | 2.97 | 5.36 | 1.83 | 1.66 | 16.158 |

The percentage of processor time is the percentage of time the processor spends executing an idle thread and then subtracting the value from 100%. This is more exact than simply reading what the processor utilization is at, at any one time. This was measuring how much of the processor is being used by each thread that is running. The collected data was separated out onto our lower-resourced databases and then again for our final 2 systems with production overlaid. It might appear that the higher-resourced cloud database was running close to 100% all the time, however this was because we collected data consistently for the entire time period and then placed all this data on one small graph. When you refer to the chart above you can see that as we added resources, the amount of total processor utilized became minimal.

### (7) Disk Reads/sec

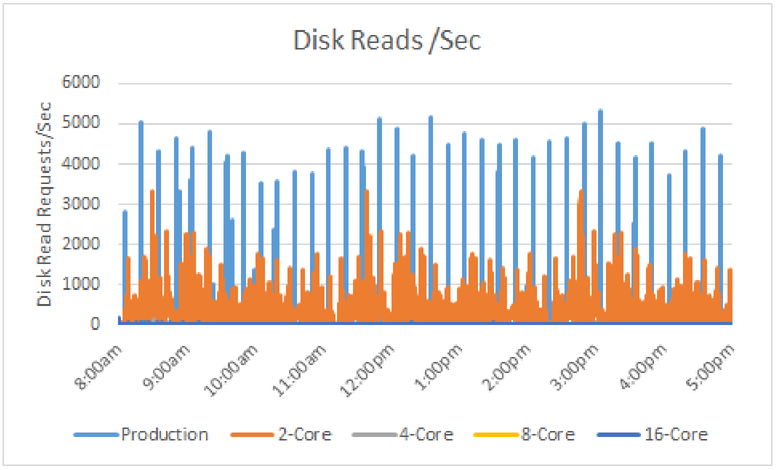


Figure 12. Amount of disk reads per second for all five iterations.

Table 11. Average disk read requests per second.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 289.85 | 0.55 | 50.1 | 0.15 | 0.096 | 68.1 |

Performance Monitor captures the total amount of individual IO requests over the period of a second against the disk. Here we saw that in production the amount of reads requests against the disk are fewer, but each are typically large in volume. These queries, outside of the most expensive ones are not replicated in our 2-Core environment, thus we saw that the total reads for our environment are much lower. However, on average there are more reads perpetrated in our lesser system; why is that? We wanted it that way. If you refer to Section C regarding NEWID, we were attempting to generate random queries so that we could avoid every page being simply cached in memory. This chart showed us that 2GB of RAM was insufficient and thus the database had to page to disk when it ran out of memory. After the resources were increased to 4GB of RAM this was no longer an issue and we see little disk utilization in the following trials.

**(8) Disk Read Bytes/sec**

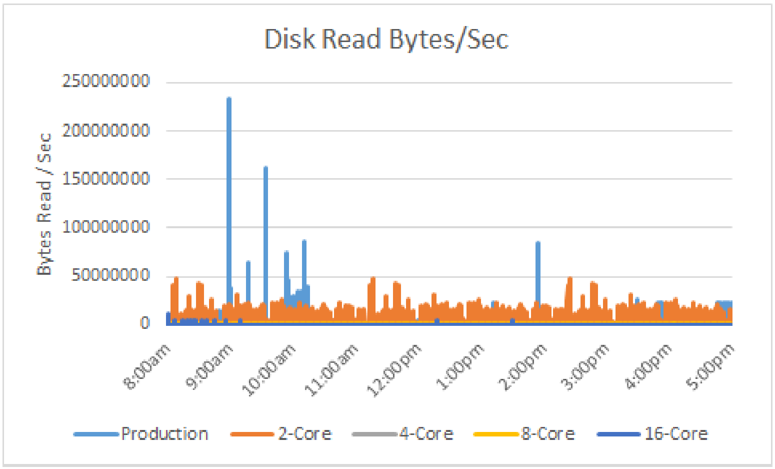


Figure 13. Bytes retrieved from the disk per each read.

Table 12. Average number of bytes retrieved per request for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 3189375 | 57063 | 1125368 | 10015 | 6190 | 866189 |

Disk read bytes/sec returns the number of bytes retrieved from the disk over the period in which it is queried. We saw the amount of bytes queried was rather consistent, this was because we queried a random percentage of the queries from production. Our production system had a few larger reads, the biggest being 230kb in size. As we stated before, our lowest resourced machine would page to disk constantly under the query load. After resources were increased to 4 CPUs and above, we no longer saw much disk reading as the data set would become more and more complete in memory.

**(9) Disk Writes/Sec**

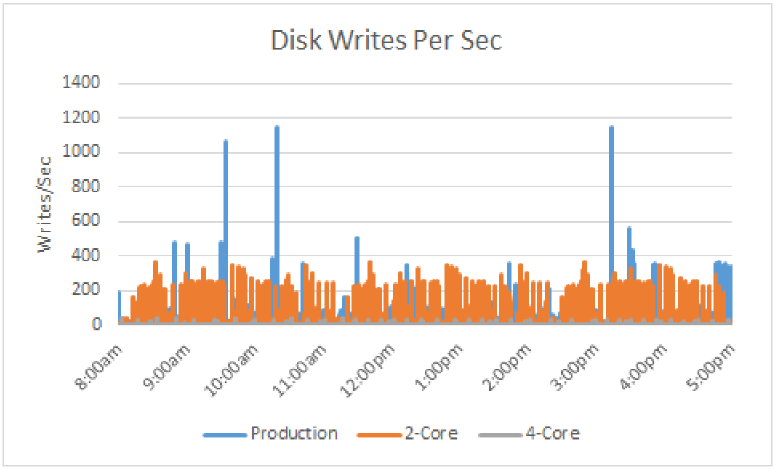


Figure 14. Disk writes per second for production overlaid with our two smaller systems.

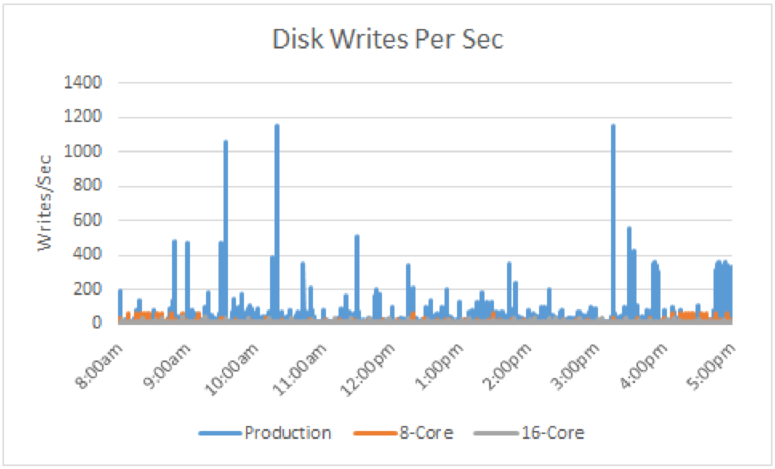


Figure 15. The disk writes per second overlaid with production and our two final higher-resourced machines.

Table 13. Average disk write requested per second for each system configuration.

| | 2-Core | 4-Core | Prod | 8-Core | 16-Core | Total |
|---|---|---|---|---|---|---|
| Average | 9.57 | 1.53 | 6.56 | 0.62 | 0.58 | 3.772 |

Disk write/sec captures the total number of individual disk write requests completed over a period of a second. We expected only a minor amount of writes against the lower and upper resourced cloud servers. We also expected them to be exactly the same amount of writes. However, here we saw just the opposite. As we hypothesized in the previous two graphs, we would have evidence of paging to disk if we saw inconsistencies in the reads. Here we had more writes occurring in our 2-Core server. After we determined that there was enough queueing in our lesser-resourced server to cause less throughput, we expected that there would be less writes as well. Little can be gleaned in the performance differences between our higher-resourced servers and production because the memory was never severely impacted.

# V. CONCLUSIONS AND FUTURE WORK

We present our conclusions in this chapter based on the data collected and analyzed from the SQL Server traces against our production and cloud servers. The cloud-based system needs to meet similar query completion time and CPU utilization figures as production and must demonstrate increased throughput when scaled upward for us to recommend cloud deployment of the PYTHON database. In our hypothesis we predicted that the cloud-based server would perform close to or slower than the standalone server, while using more resources and when scaled upwards, would slightly increase its performance

## A. CONCLUSIONS

The major conclusions from our experiment are as follows.

**(1) With similar levels of processing and storage capacity, the cloud-based system's average query completion time was similar to production.**

The average query completion time is a dependable demonstration of a database's throughput. This is even truer in our experiment due to the fact that we created synthetic traffic, subjecting each VM configuration to the same amount of load. In our standalone system the average query completion time is 136746 microseconds. On our cloud-based VM with comparable hardware, the average was only 62 milliseconds slower. The database that we based this experiment is a reporting database, making the 62 ms difference reasonable because of the large variety of complex queries that get executed on a near-constant schedule. The average time it takes to complete a query in production is 0.136 seconds, and in our experiment 0.198 seconds.

**(2) The cloud-based server utilized resources more efficiently.**

This test was a big win for the cloud-based VM. The VM performed similarly to production and on average utilized less resource, with only 2.97% used of its 4 cores on average compared to the production system’s average of 5.36%. There was a similar trend with respect to the processor queue length. The cloud-based VM averaged .089 processes queued per second, while the production machine received .11 processes. A large concern for anyone migrating to a cloud-based hosting environment should be efficiency.  Even more so in a private-cloud where one has to personally host and procure all the systems themselves. The results of this experiment show that migrating databases similar to the IMS of the Naval Postgraduate School, namely data warehouses and reporting databases, lend well to being hosted in a private-cloud.

**(3) The performance of the cloud-based system increased significantly when granted sufficient additional resources.**

One of the primary selling points of a cloud-based system is the ease of resource allocation, either in response to demand, or pro-actively in anticipation of a large increase in traffic. We doubled and then quadrupled the resources allocated to determine if this would affect performance without an increased workload. With our VM increased to 8 CPU cores, the average query completion time dropped to 124478 microseconds, making it .012 seconds faster than the production system. When increased to 16 CPU cores we would see a dramatic increase in query completion time, as the average time dropped to 29171 microseconds.

The data collected from our trial proves that reporting-style databases are well suited for deployment to a private-cloud environment. Already, deploying in a private-cloud environment ensures that the DOD can administer the security and address the reliability of its servers. We have now shown that when deploying reporting-type databases, such as data-warehouses, the performance and scalability advertised by cloud services can be attained.

## B. FUTURE WORK

If I were to perform this experiment again, I would increase the types of databases used and extend the type of cloud environments surveyed to include public-cloud offerings.

The work performed in this thesis can be used as a basis in evaluating other DOD reporting-style database applications for their suitability for private-cloud hosting. If we were to extend this thesis, then including transactional databases would expand the possible application of our findings. Using a transactional database instead of a reporting-style database would mean we would have a larger percentage of write requests to our hard drives, thus allowing us to evaluate their performance. It also would place a much higher demand on memory utilization and page life expectancy, which was an area of this thesis not fully explored due to the type of databases used.

To further extend the application of our thesis, we would want to perform our experiment in a public-cloud environment. Here we could evaluate across multiple hosts and the levels of performance offerings available. Additionally, metrics such as latency due to hosting location will play a big factor in suitability. With this data we could evaluate across multiple arenas, including public, private and standalone environments.

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES

[1] American Forces Press Service. (2014, April 15). DOD releases report on estimated sequestration impacts [Online]. Available: http://www.defense.gov/news/newsarticle.aspx?id=122065

[2] S. Vogel. (2013, May 29). Defense workers receive furlough notices [Online]. Available: http://www.washingtonpost.com/blogs/federal-eye/wp/2013/05/29/defense-workers-begin-receiving-furlough-notices/

[3] T. Lathe. (2014, July 9). Law enforcement access to data in the cloud [Online]. Available: http://www.rackspace.com/blog/law-enforcement-access-to-data-in-the-cloud/

[4] D. Goodin. (2015, May 13). Extremely serious virtual machine bug threatens cloud providers everywhere [Online]. Available: http://arstechnica.com/security/2015/05/extremely-serious-virtual-machine-bug-threatens-cloud-providers-everywhere/

[5] L. C. Williams. (2014, November, 26). Happy Black Friday, your credit card could very likely be stolen again [Online]. Available: http://thinkprogress.org/economy/2014/11/26/3597288/target-breach-anniversary/

[6] Miniwatts Marketing Group. (2014, December 1). Internet growth statistics [Online]. Available: http://www.internetworldstats.com/emarketing.htm

[7] Institut Numêrique. (2013, June 18). Chapter 1 : The concepts of cloud computing [Online]. Available: http://www.institut-numerique.org/chapitre-1-les-concepts-du-cloud-computing-51c0279ca534a

[8] S. Basu, "Productivity growth in the 1990s: Technology, utilization, or adjustment?," NBER Working Paper No. 8359, pp. 2–3, July 2001.

[9] D. Law. (2013, February 18). Outlook.com leaves preview as the world's fastest growing email service going from 0 to 60 million in just 6 months [Online]. Available: http://blogs.office.com/2013/02/18/outlook-com-leaves-preview-as-the-worlds-fastest-growing-email-service-going-from-0-to-60-million-in-just-6-months/

[10] Google. (2015, March 23). Company history [Online]. Available: http://www.google.com/about/company/history/

[11] C. Janssen. (2014, December 22). Elastic computing [Online]. Available: http://www.techopedia.com/definition/26598/elastic-computing-ec

[12] J. McKendrick (2014, December 23). Without PaaS, Docker is just a bunch of containers [Online]. Available: http://www.zdnet.com/article/paas-and-docker/

[13] Wikipedia (2015, June 09). Cloud computing [Online]. Available: http://en.wikipedia.org/wiki/Cloud_computing

[14] M. Rouse (2015, January 1). Infrastructure as a Service (IaaS) [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS

[15] R. Sheldon (2014, March). Database as a Service [Online]. Available: http://whatis.techtarget.com/definition/Database-as-a-Service-DBaaS

[16] J. Sanders (2014, July 1). Hybrid Cloud: What it is, why it matters [Online]. Available: http://www.zdnet.com/article/hybrid-cloud-what-it-is-why-it-matters/

[17] Microsoft (2015). What is Microsoft Azure [Online]. Available: http://azure.microsoft.com/en-us/overview/what-is-azure/

[18] Amazon (2015). Amazon EC2 [Online]. Available: http://aws.amazon.com/ec2/

[19] J. McKendrick (2013, July 21). 5 Benefits of cloud computing you aren't likely to see in a sales brochure [Online]. Available: http://www.forbes.com/sites/joemckendrick/2013/07/21/5-benefits-of-cloud-computing-you-arent-likely-to-see-in-a-sales-brochure/

[20] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing, pages 193–204, Indianapolis, USA, 2010

[21] R. Miller (2012, March 14). Estimate Amazon cloud backed by 450000 servers [Online]. Available: http://www.datacenterknowledge.com/archives/2012/03/14/estimate-amazon-cloud-backed-by-450000-servers/

[22] D. J. Power (2013, October 12). What is ACID and BASE in database theory [Online]. Available: http://dssresources.com/faq/index.php?action=artikel&id=281

[23] Microsoft (2007, April 25). Performance and reliability monitoring getting started guide for Windows Server 2008 [Online]. Available: https://technet.microsoft.com/en-us/library/cc771692.aspx

[24] Microsoft (2015). SQL server profiler [Online]. Available: https://msdn.microsoft.com/en-us/ms181091.aspx

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California